Optimisation of Convolution of Multiple Different Sized Filters in SKA Pulsar Search Engine

Haomiao Wang
PARC Lab
University of Auckland
Auckland, NewZealand
hwan938@aucklanduni.ac.nz

Ben Stappers
Jodrell Bank Centre for Astrophysics
University of Manchester
Manchester, UK
ben.stappers@manchester.ac.uk

Prabu Thiagaraj
Raman Research Institute
University of Manchester
Manchester, UK
prabuthiagaraj@gmail.com

Oliver Sinnen
PARC Lab
University of Auckland
Auckland, NewZealand
o.sinnen@auckland.ac.nz

***Abstract*—Pulsar search is one of the main tasks for the Square Kilometre Array (SKA) central signal processor (CSP) sub-element. Because most of the pulsar details are unknown, many pulsar search approaches are employed. The main compute-intensive application of the pulsar search modules is the matched filter group, which convolves the input signals with a group of filters. High-performance designs on FPGAs have been proposed that can process multiple large filters efficiently. But given that in many applications, including the here targeted pulsar search, filters have different sizes, there is a high potential for optimisation. This paper investigates the optimisation of general matched filtering design for the SKA pulsar search engine. The influence of changing the number of filters and the difference in sizes is analysed. The general implementations in time-domain (TD) and frequency-domain (FD) are optimised, employing the longest processing time (LPT) first rule to distribute filter templates across filter processing pipelines. The proposed design is employed to implement the matched filter groups in two SKA pulsar modules. The results show that the optimisation can provide up to 2.1x speedup in TD and 1.2x speedup in FD.**

***Keywords*-FPGA; OpenCL; FIR filter; Pulsar search;**

I. Introduction

The scale of the Square Kilometre Array (SKA) is tens of times larger than other radio telescope arrays, and so is the number of received signals. This is a big challenge for the SKA central signal processor (CSP). One of the main elements in the CSP is the pulsar search engine (PSS) that employs several approaches to search for different types of pulsars. Since most of the details of a pulsar are unknown, the PSS has to search for a wide range of values for each parameter. The matched filtering technique is an efficient approach to recover the polluted signals and it is widely employed in pulsar search modules. For the SKA1-MID pulsar search engine, it appears in many modules such as single pulsar detection (SPDT), Fourier domain acceleration search (FDAS), and folding and optimisation (FLDO). However, the employed matched filter groups are not the same such as the number of filters and the filter tap incremental.

The main feature of the SKA pulsar search module is that the input data size is hundreds of times larger than the output data size, and the input data cannot be stored in on-chip (FPGA) memory. For example, the input signals sizes to SPDT, TDAS, and FDAS are millions of points. However, the output signals are all a list of pulsar candidates, whose sizes are less than 1% of those of input signals. Because of the huge amount of workload and restricting time limitation, the needed throughput is high, and the high-end acceleration devices are required. This makes the high-end FPGA an ideal hardware accelerator compared to other devices.

In this research, we investigate the FPGA-based optimisation of matched filtering design with different sizes. The main contributions are as follows:

- Analysis and modeling for various sized matched filter groups;
- Design of an optimisation approach for the general FPGA-based matched filtering implementation using load balance algorithm;
- Employing the proposed design to optimise two matched filter groups in different SKA1-MID pulsar search modules and their evaluation.

The rest of the paper is organized as follows. Section II gives background on matched filtering in radio astronomy and FPGA as an accelerator. Section III discusses matched filtering, the design goals, and gives a theoretical analysis. In Section IV, the proposed designs of the matched filter group are discussed, and the load balance algorithm is employed for multiple threads. Section V presents the evaluation, and the results are discussed. Finally, the conclusions are given in Section VI.

II. Related Work

High-end FPGA accelerators are widely employed to handle large-scale computation tasks in many radio astronomy projects [6]. Hundreds of Xilinx FPGAs are installed to accelerate the correlator of the SKAMP project, and the FPGA-based accelerator appears in the FX correlator of MeerKAT as well [6]. Besides correlator, high-end FPGA platforms are employed to handle digital channelised receivers.

Regarding the matched filter technique in pulsar search, it is widely employed, and many known pulsars have been

found because of it [3]. The basic element of the matched filtering is the 1D convolution. In [5], it is evaluated on different platforms, and FPGA devices perform better than CPU and GPU when there are several hundred coefficients.

III. Matched Filtering

A. FIR Filter

A matched filter is employed to correlate a known template with unknown signals to detect the presence of the template in the unknown signals. For pulsar search, the details of a pulsar are unknown, and a group of predicted templates are employed. The signal array is convolved with N_{filter} templates, which can be presented as

$$y[i][j] = \sum_{k=1}^{N_{tap_i}} x[j-k]h[i][k],$$
$$for\, i = 1, 2, ...N_{filter}\, and\, j = 1, 2, ...N_{input}$$

where y is the filter output plane, x is the input, h is the coefficient array, and N_{tap_i} is the length of the ith filter. Regarding the input and coefficient arrays in the SKA project, all the data types are complex single-precision floating-point (SF). The lengths from filter$_1$ to filter$_{N_{filter}}$ are different, the relationship between these filter is

$$N_{tap_{i+1}} = N_{tap_i} + N_{inc}$$

$$N_{tap_{N_{filter}}} = N_{tap_1} + N_{inc}(N_{filter} - 1),$$

where we call N_{inc} the tap incremental. There are mainly three factors that determine a matched filter group: 1) N_{filter}, 2) N_{inc}, and 3) [N_{tap_1}, $N_{tap_{N_{filter}}}$]. In this research, we use $MF-(N_{filter}, N_{inc}, N_{tap_1})$ to represent a matched filter group.

For the fundamental element of the matched filter group, which is the FIR filter, it can be implemented in both time domain (TDFIR) and frequency domain (FDFIR). For the large tap TDFIR, the overlap-add (OLA) algorithm can be employed to split the FIR taps into a group of small filters. Regarding the FDFIR filter with large input array, the overlap-save (OLS) algorithm can be employed to divide the input array into a group of small input arrays [4].

B. Design Goals

For the FPGA-based acceleration, the FPGA configuration time can be over one second, and even partial reconfiguration takes tens to a hundred of milliseconds. To avoid configuration, all filters are implemented using one FPGA image (bitstream file). For the generic implementation [2], the number of taps of all filters is the same, which is $N_{tap_{N_{filter}}}$, and one design can implement all filters. If a filter is smaller than $N_{tap_{N_{filter}}}$, it is padded with zero to make its length to $N_{tap_{N_{filter}}}$. Since many filters are extended this way, a large proportion of operations are –unnecessary. In this research, we investigate the optimisation of the generic FPGA-based matched filtering implementation. The main goal is to propose an optimised design for given matched filter group and a specific FPGA.

C. Theoretical Analysis

1) Time-Domain (TD) Filtering: Based on the features in Section III-A, the total number of taps is

$$\sum_{i=1}^{N_{filter}} N_{tap_i} = N_{filter}N_{tap_1} + \frac{N_{inc}N_{filter}(N_{filter}-1)}{2}.$$

For a generic matched filter group implementation, the filter lengths are padded to be the same as $N_{tap_{N_{filter}}}$, and the sum of all taps is $N_{tap_{N_{filter}}}N_{filter}$. It can be seen that the difference of these two values is $N_{inc}N_{filter}(N_{filter}-1)/2$, which is related to N_{inc} and N_{filter}, and not affected by . As the number of FIR filters increases, the ratio of saved taps approaches 0.5, which means up to 50% of operations can be saved.

Regarding the OLA algorithm, each filter is split into a group of small filters, and each length is $N_{OLA-tap}$, whose value has to be smaller than $N_{tap_{N_{filter}}}$. By employing the OLA algorithm, the total number of taps is reduced from $N_{tap_{N_{filter}}}N_{filter}$ to $\sum_{i=1}^{N_{filter}} \left\lceil \frac{N_{tap_i}}{N_{OLA-tap}} \right\rceil N_{OLA-tap}$. The smaller the $N_{OLA-tap}$, the larger the ratio of saved operations. When $N_{OLA-tap}$ is 1, the ratio of saved taps is the highest, but might not be achievable due to bandwidth and resource problems.

2) Fourier-Domain (FD) Filtering: For the straightforward FDFIR, the length of an FIR filter does not influence the execution latency, which is only related to the number of filters. In the OLS algorithm, the input array is split into a group of chunks, and the length of each chunk is N_{OLS-FT}, which is the Fourier transform length of the OLS algorithm as well. Each chunk is overlapped with its neighbourhood chunks, and the length of this overlap is identical to the number of filter taps.

For any FIR filter i, the number of chunks is $N_{OLS-chunk_{N_i}} = \left\lceil \frac{N_{input}}{N_{OLS-FT}-N_{tap_i}} \right\rceil$, and the total number of saved chunks compared with the generic filter implementation is

$$N_{filter}N_{OLS-chunk_{N_{filter}}} - \sum_{i=1}^{N_{filter}} \left\lceil \frac{N_{input}}{N_{OLS-FT}-N_{tap_i}} \right\rceil.$$

For all N_{filter} filters, $N_{tap_{N_{filter}}}$ has to be smaller than N_{OLS-FT}. Figure 1 presents the saved chunks compared to a generic FDFIR implementation in processing 2^{30} input points. Figure 1 (left) shows the ratio of saved chunks as a colour map, depending on the filter incremental (N_{inc}, y-axis) and the number of filters (N_{filter}, x-axis). To investigate the relationship between the saved ratio and $N_{tap_{N_{filter}}}$. All points that apply $(\left\lceil \frac{N_{filter}}{10} \right\rceil, N_{filter})$ in Figure 1 (right) are picked, which is the

red dotted line in Figure 1 (left). This chart depicts the relationship between the saved ratio (y-axis) and

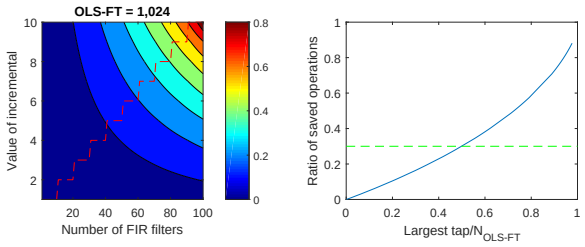

Figure 1. Saved operations of changing the value of incremental and the number of filters applying OLS algorithm

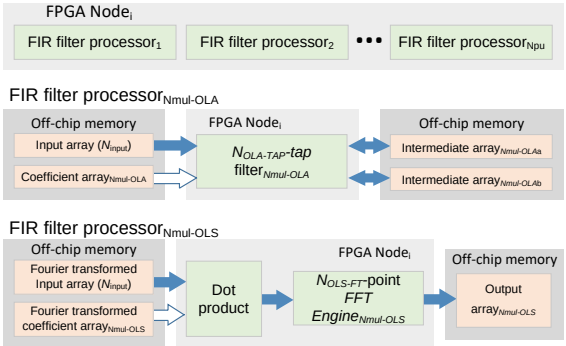

Figure 2. Architecture of a single processor in multiple OLA-TD and multiple OLS-FD implementations on one FPGA node

$N_{tap_{N_{filter}}}/N_{OLS-FT}$. If $N_{tap_{N_{filter}}}$ equals to $\frac{N_{OLS-FT}}{2}$, up to 30% (the point of intersection with the green dotted line) operations can be saved.

IV. Optimisation and Implementation

A. Implementation

It is assumed that the size of input signals is larger than the device on-chip memory size and it has to be stored in off-chip memory during processing. Regarding the coefficient array, it can be stored in both off-chip memory and on-chip memory based on its size. The details of the architecture are depicted in Figure 2, where N_{pu} is the number of processing units. It equals $N_{mul-OLA}$ when employing OLA-TD and equals to $N_{mul-OLS}$ when employing OLS-FD.

1) OLA-TD : For the OLA-TD algorithm, multiple FIR filters $N_{mul-OLA}$, whose lengths are all $N_{OLA-tap}$, are implemented in one FPGA image. The number of DSP blocks on an FPGA $N_{FPGA-DSP}$ and off-chip memory bandwidth are two main factors that affect the value of $N_{mul-OLA}$ when the data types of input signals and coefficient arrays are floating-point.

The details of one FIR filter processor using OLA-TD algorithm are depicted in Figure 2. Each processor needs two buffers in off-chip memory to store the intermediate arrays, whose length is $2N_{input}$. Based on the analysis in Section III-C, the smaller the $N_{OLA-tap}$, the less the number of unnecessary operations and used DSP blocks. However, the decrease of $N_{OLA-tap}$ leads to the increase of

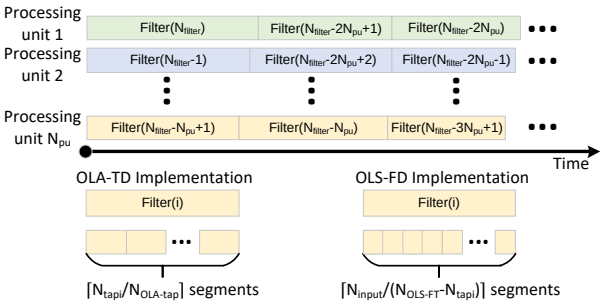

Figure 3. Processing order of matched filter group by employing the LPT rule

$N_{mul-OLA}$. This will make the off-chip memory bandwidth become the main factor that limits the performance.

2) OLS-FD: Regarding the OLS-FD algorithm, we investigate two main factors 1) the length of the overlapped part (N_{tap_i}) and 2) the number of input chunks for the FIR filter ($N_{OLS-chunk_i} = \left\lceil \frac{N_{input}}{N_{OLS-FT}-N_{tap_i}} \right\rceil$). These two factors for all filters in the matched filter group are not the same and these changes can be implemented in the host programs. The overlapped part length can be adjusted depending on the number of taps before sending to the FPGA, e.g. using `memcpy` function in C++. Regarding the second factor, it can be adjusted by changing the number of iterations of the outer `for` loop.

For the FPGA-based implementation, the structure of a single processor is given in Figure 2. It consists of two parts: the dot product and N_{OLS-FT}-point FFT engine. It has two significant differences from the OLA-TD implementation: 1) the input signals and coefficient arrays have to be Fourier transformed before processing, and 2) the output from FFT engine can be sent to off-chip memory directly without loading and adding the output from the previous block.

B. Load Balancing

Based on the above discussion, multiple filters N_{pu} can be put into one FPGA image. For matched filtering, the longest processing time rule (LPT) [1], which is a 4/3 approximation algorithm, can be employed to balance the execution latency of each thread, i.e. implemented hardware filter. For any filter i in the matched filter group, where $i \neq N_{filter}$, $N_{tap_i} < N_{tap_{i+1}}$, so the lengths from the last filter to the first filter are already in decreasing order. The process order by employing LPT on N_{pu} processing units is depicted in Figure 3.

For OLA-TD, the execution time of the ith filter contains $\left\lceil \frac{N_{tap_i}}{N_{OLA-tap}} \right\rceil$ segments, and the latency of each segment is about the same. For OLS-FD, $\left\lceil \frac{N_{input}}{N_{OLS-FT}-N_{tap_i}} \right\rceil$ segments with the same latency form the execution latency of the ith filter. When weighting based on lanchtimes ornumbers of chunks, the processing order is not rondrobin.

Table I
PROCESSING UNITS FOR ALGORITHMS ON A SPECIFIC DEVICE

Device		OLA-TD-($N_{OLA-tap}$)				OLS-FD-(N_{OLS-FT})		
		16	32	64	128	1024	2048	4096
S5	N_{pu}	4	2	1	–	4	3	3
	Logic	68%	64%	50%	–	85%	68%	72%
	DSP	100%	100%	100%	–	100%	89%	90%
	RAM	32%	26%	20%	–	74%	88%	92%
	f_{max}	223.6	226.6	232.5	–	217.0	205.7	177.2
A10	N_{pu}	15	7	3	1	4	4	3
	Logic	47%	33%	25%	21%	28%	33%	26%
	DSP	94%	88%	76%	51%	38%	43%	36%
	RAM	64%	35%	23%	17%	58%	60%	73%
	f_{max}	153.0	191.0	213.6	258.9	203.5	165.4	158.0

V. EXPERIMENTS AND RESULTS

A. Experimental Setup

Two types of FPGA devices are adopted in this research and they are used to evaluate the performance of the same design. We used two FPGA acceleration cards, aTerasic DE5 board with Stratix V, referred to as **S5**, and a Nallatech 385A board with Arria 10, referred to as **A10**. The S5 and A10 cards are connected to the host processor through the PCI Express (PCIe) bus.

For different FPGA series and devices, the high-level synthesis approach OpenCL is employed for fast prototyping. All Intel FPGA-based OpenCL kernels are compiled using the Intel OpenCL offline compiler (AOCL) version 16.0.0.211.

Two matched filter groups from two pulsar search modules are evaluated in this section:

- FDAS: $MF_{FDAS} - (42,\ 10,\ 11) \times 2$, $N_{input} = 2^{21}$
- FLDO: $MF_{FLDO} - (64,\ 1,\ 1) \times 1$, $N_{input} = 2^{10}$.

B. Resource Usage

The OLA-TD-($N_{OLA-tap}$) and OLS-FD-(N_{OLS-FT}) designs are implemented on both S5 and A10, and the maximum numbers of parallelised processing units are given in Table I. It can be seen that the dominant resource for OLA-TD-($N_{OLA-tap}$) is DSP blocks. Regarding OLS-FD-(N_{OLS-FT}), the number of RAM blocks becomes the main factor that restricts N_{pu}.

Although the compiler versions for both devices are the same, the board support package (BSP) are provided by different vendors. For the same kernel code on S5 and A10, the kernel frequency and the resource costs vary.

C. Latency Evaluation

Since the Intel FPGA-based OpenCL kernel frequency is decided by the compiler and cannot be set manually, we compare the number of clock cycles, calculated from the kernel frequency and the execution latency which are given in the kernel profile document after execution.

For the generic implementation, the tasks are 43x 421-tap filters and 64x 64-tap filters, respectively. The best-performed methods in TD and FD for both S5 and A10 are OLA-TD-(64) and OLS-FD-(2048). Table II gives the speedups of employing our proposed LPT algorithm when comparing the execution latencies to the generic implementation. The LPT approach can achieve up to a 2.14x speedup in TD and 1.20x speedup in FD. The best performance is achieved on A10, which are $137ms$ using OLS-FD (FDAS) and $0.10ms$ using OLA-TD (FLDO).

Table II
SPEEDUP OF EMPLOYING LPT ALGORITHM

Device	MF	OLA-TD-($N_{OLA-tap}$)				OLS-FD-(N_{OLS-FT})		
		16	32	64	128	1024	2048	4096
S5	FDAS	**1.98**	1.95	1.83	–	1.16	1.13	0.35
	FLDO	1.59	1.33	1	–	**1.20**	1.02	0.30
A10	FDAS	**2.14**	2.00	1.87	1.12	0.93	**1.12**	0.66
	FLDO	1.45	1.36	1	0.35	0.94	1.01	0.63

VI. CONCLUSIONS

This paper presented the detailed optimisation of multiple complex floating-point filters of different sizes. Various sized matched filter groups were analysed in time-domain and frequency-domain using the overlap-add algorithm and the overlap-save algorithm. Both OLA-TD and OLS-FD processing can improved in comparison to the generic implementation that extends the lengths of all filters to the largest filter in the matched filter group. Two matched filter groups from different SKA pulsar search modules were evaluated. The execution latencies showed that the optimised designs with the LPT algorithm are up to 2.1x and 1.2x faster than the generic implementation, in TD and FD respectively. It should be noted, that for a specific matched filter group on different devices, the processing domain and the best performance parameters of the employed algorithms are different.

REFERENCES

[1] Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. SIAM journal on Applied Mathematics, 17(2), 416-429.

[2] Wang, H., Zhang, M., Thiagaraj, P., & Sinnen, O. (2016, December). FPGA-based acceleration of FDAS module using OpenCL. In Field-Programmable Technology (FPT), 2016 International Conference on (pp. 53-60). IEEE.

[3] Ransom, S. M., Eikenberry, S. S., & Middleditch, J. (2002). Fourier techniques for very long astrophysical time-series analysis. The Astronomical Journal, 124(3), 1788.

[4] Pavel, K., & David, S. (2013). Algorithms for efficient computation of convolution. In Design and Architectures for Digital Signal Processing. InTech.

[5] Fowers, J., Brown, G., Wernsing, J., & Stitt, G. (2013). A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors. ACM Transactions on Architecture and Code Optimization (TACO), 9(4), 25.

[6] Parsons, A., Werthimer, D., Backer, D., Bastian, T., Bower, G., Brisken, W., ... & Greenhill, L. (2009, March). Digital Instrumentation for the Radio Astronomy Community. In astro2010: The Astronomy and Astrophysics Decadal Survey (Vol. 2010).